+--------------------+
| CS 140 |
| PROJECT 1: THREADS |
| DESIGN DOCUMENT |
+--------------------+

---- GROUP ----

>> Fill in the names and email addresses of your group members.

Yochana Yellaboyana <yochanay@buffalo.edu>
Tony Jason Anga <tonyjaso@buffalo.edu>
Nikhila Doddapaneni <nikhilad@buffalo.edu>

---- PRELIMINARIES ----

>> If you have any preliminary comments on your submission, notes for the
>> TAs, or extra credit, please give them here.

>> Please cite any offline or online sources you consulted while
>> preparing your submission, other than the Pintos documentation, course
>> text, lecture notes, and course staff.

ALARM CLOCK

---- DATA STRUCTURES ----

>> A1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less.

```
/ Used to store ticks count - amount of time thread should sleep /
int ticks;
/Store the threads put to sleep/
struct list sleep_list;
```

---- ALGORITHMS ----

>> A2: Briefly describe what happens in a call to timer_sleep(),
>> including the effects of the timer interrupt handler.

timer_sleep()

1. Suspends execution for X ticks
2. Add thread to sleep_list
3. Stop thread from execution
4. After x ticks bring back the thread to ready queue

Interrupt handler:
1. Iterate through the list to see if the thread needs to awake
2. If yes remove from sleep list
3. Reset ticks to original value
4. Reset thread to original values

>> A3: What steps are taken to minimize the amount of time spent in
>> the timer interrupt handler?
Sleep list is a sorted array based on its ticks. Current ticks are counted and if greater than the ticks of a thread wake it up. No need to check next as it is sorted.

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call
>> timer_sleep() simultaneously?

>> A5: How are race conditions avoided when a timer interrupt occurs
>> during a call to timer_sleep()?
Interrupts are disabled

---- RATIONALE ----

>> A6: Why did you choose this design? In what ways is it superior to
>> another design you considered?
Adding threads to list is one of the basic methods considered. Threads that call timer_sleep can simply be added to sleep_list in ascending order and can be tracked easily. Once a ceratin number of ticks pass by we can iterate through the list fast as it is already sorted and put it back to ready queue if the current ticks is greater than the ticks of the thread in list. No other design was considered.

PRIORITY SCHEDULING
==========

---- DATA STRUCTURES ----

>> B1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less.

>> B2: Explain the data structure used to track priority donation.
>> Use ASCII art to diagram a nested donation. (Alternately, submit a
>> .png file.)

---- ALGORITHMS ----

>> B3: How do you ensure that the highest priority thread waiting for
>> a lock, semaphore, or condition variable wakes up first?
From the list of threads which are waiting for lock or semaphore we search for highest priority and move it to ready list.
>> B4: Describe the sequence of events when a call to lock_acquire()
>> causes a priority donation. How is nested donation handled?

>> B5: Describe the sequence of events when lock_release() is called
>> on a lock that a higher-priority thread is waiting for.

---- SYNCHRONIZATION ----

>> B6: Describe a potential race in thread_set_priority() and explain
>> how your implementation avoids it. Can you use a lock to avoid
>> this race?

---- RATIONALE ----

>> B7: Why did you choose this design? In what ways is it superior to
>> another design you considered?

ADVANCED SCHEDULER

---- DATA STRUCTURES ----

>> C1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less.

```
struct thread
  int nice;                   /* Thread nice value */
  int recent_cpu;                 /* Thread recent CPU */

In thread.h
/* Nice value boundary */
#define NICE_MIN -20
#define NICE_DEFAULT 0
#define NICE_MAX 20
```

In thread.c
static int load_avg; /* load_avg for BSD scheduling. Fixed-point number */

---- ALGORITHMS ----

>> C2: Suppose threads A, B, and C have nice values 0, 1, and 2. Each
>> has a recent_cpu value of 0. Fill in the table below showing the
>> scheduling decision and the priority and recent_cpu values for each
>> thread after each given number of timer ticks:
Recent_cpu is calculated using the below formula

recent_cpu = (2*load_avg)/(2*load_avg + 1) * recent_cpu + nice,

load_avg = (59/60)*load_avg + (1/60)*ready_threads

	recent_cpu		priority			
timer ticks	A	B	C	A	B	C	thread to run	Note
0	0	1	2	63	61	59	A	
4	4	1	2	62	61	59	A	
8	7	2	4	61	61	58	B	Round robin
12	6	6	6	61	59	58	A	
16	9	6	7	60	59	57	A	
20	12	6	8	60	59	57	A	
24	15	6	9	59	59	57	B	Round robin
28	14	10	10	59	58	57	A	
32	16	10	11	58	58	56	B	Round robin
36	15	14	12	59	57	56	A	

>> C3: Did any ambiguities in the scheduler specification make values
>> in the table uncertain? If so, what rule did you use to resolve
>> them? Does this match the behavior of your scheduler?
As it has not yet been implemented the values are calculated using the formula, but recent_cpu calculation is not considered so if we consider load_avg, recent_cpu calculations and priority calculation of threads, so we may notice few ambiguities in the scheduler.
>> C4: How is the way you divided the cost of scheduling between code
>> inside and outside interrupt context likely to affect performance?

---- RATIONALE ----

>> C5: Briefly critique your design, pointing out advantages and

>> disadvantages in your design choices. If you were to have extra
>> time to work on this part of the project, how might you choose to
>> refine or improve your design?

>> C6: The assignment explains arithmetic for fixed-point math in
>> detail, but it leaves it open to you to implement it. Why did you
>> decide to implement it the way you did? If you created an
>> abstraction layer for fixed-point math, that is, an abstract data
>> type and/or a set of functions or macros to manipulate fixed-point
>> numbers, why did you do so? If not, why not?

SURVEY QUESTIONS

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want--these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

>> In your opinion, was this assignment, or any one of the three problems
>> in it, too easy or too hard? Did it take too long or too little time?

>> Did you find that working on a particular part of the assignment gave
>> you greater insight into some aspect of OS design?

>> Is there some particular fact or hint we should give students in
>> future quarters to help them solve the problems? Conversely, did you
>> find any of our guidance to be misleading?

>> Do you have any suggestions for the TAs to more effectively assist
>> students, either for future quarters or the remaining projects?

>> Any other comments?